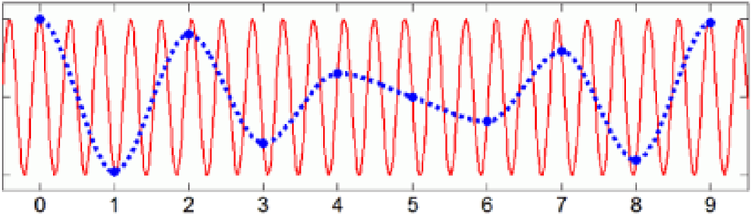
Verifying Faust in Coq

Progress report

Emilio J. Gallego Arias, Pierre Jouvelot, Olivier Hermant, Arnaud Spiwack

MINES ParisTech, PSL Research University, France

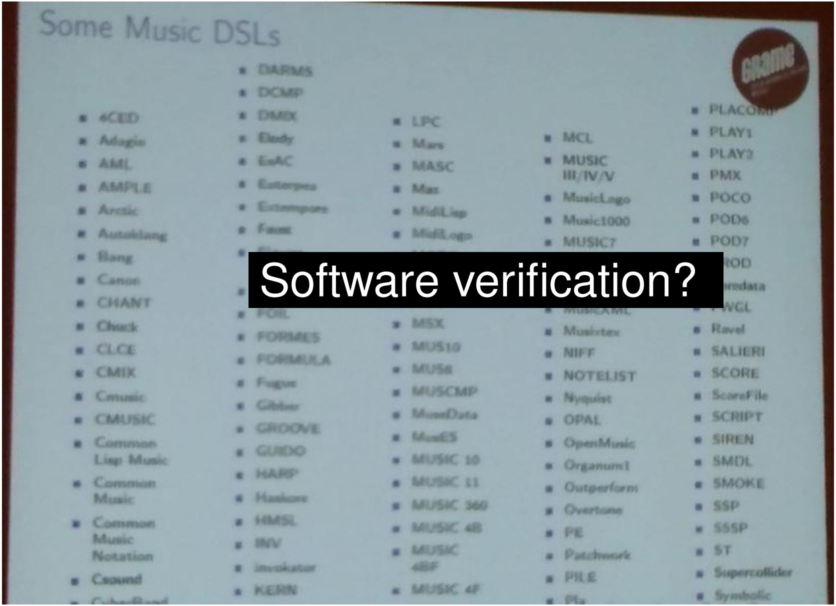
CoqPL 2015



Music and PL?

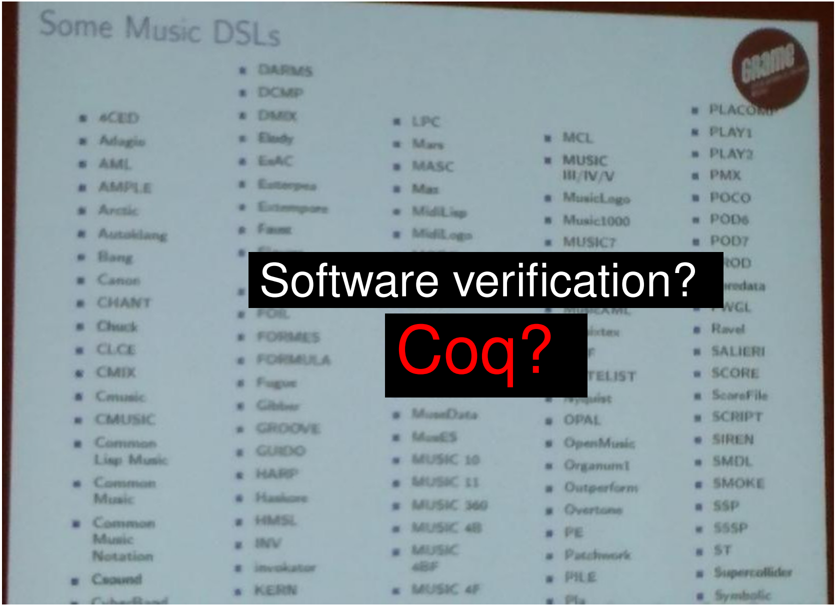

Some Music DSLs
DARMS
DCMP
DMIX
EaAC
Faust
FORMES
FORMULA
GROOVE
GUIDO
HARP
HMSL
INV
KERN
AML
AMPLE
Arctic
Bang
Canon
CHANT
Chuck
CLCE
CMIX
Cmusic
CMUSIC
Common Lisp Music
Common Music
Common Music Notation
Csound
LPC
MASC
Max
MidiLisp
MidiLogo
MODE
MOM
MSX
MUS10
MUS8
MUSCMP
MuseData
MusES
MUSIC 10
MUSIC 11
MUSIC 360
MUSIC 4B
MUSIC 4BF
MUSIC 4F
MCL
MUSIC III/IV/V
MusicLogo
Music1000
MUSIC7
Musictex
MUSIGOL
MusicXML
Musixtex
NIFF
NOTELIST
Nyquist
OPAL
OpenMusic
Organum1
Outperform
Overtone
PE
Patchwork
PILE
PLACOMP
PLAY1
PLAY2
PMX
POCO
POD6
POD7
PROD
Puredata
PWGL
Ravel
SALIERI
SCORE
ScoreFile
SCRIPT
SIREN
SMDL
SMOKE
SSP
SSSP
ST
Supercollider
Symbolic

Music and PL?



Software verification?

Music and PL?



Software verification?

Coq?

Faust

- Functional PL for digital signal processing.
- Synchronous paradigm, geared towards audio.
- Programs: circuits/block diagrams + feedbacks.
- Semantics: streams of samples.
- *Efficiency is crucial*.
- Created in 2000 by Yann Orlarey et al. at GRAME.
- Mature, compiles to more than 14 platforms.

Faust’s Ecosystem

Users:

- Grame: Multiple projects, main developer.
- Stanford: Class/books on signal processing, STK instrument toolkit, Faust2android, Mephisto…
- Ircam: Acoustic libraries, effects libraries,…
- Guitarix, moForte guitar, etc...

Faust's Ecosystem

Users:

- Grame: Multiple projects, main developer.
- Stanford: Class/books on signal processing, STK instrument toolkit, Faust2android, Mephisto...
- Ircam: Acoustic libraries, effects libraries,...
- Guitarix, moForte guitar, etc...

It has its market! Much easier than dwelling into C.

Faust's Ecosystem

Users:

- Grame: Multiple projects, main developer.
- Stanford: Class/books on signal processing, STK instrument toolkit, Faust2android, Mephisto...
- Ircam: Acoustic libraries, effects libraries,...
- Guitarix, moForte guitar, etc...

It has its market! Much easier than dwelling into C.

Recent Events:

- Faust day at Stanford happened yesterday.
- Ongoing Faust program competition (€2,000 in prices).
- FEEVER project :)

Syntax and Well-Formedness

$$\textsc{Term}\ \frac{}{\vdash ! : 1 \to 0} \qquad \textsc{Id}\ \frac{}{\vdash \_ : 1 \to 1}$$

$$\textsc{Par}\ \frac{\vdash f_1 : i_1 \to o_1 \qquad \cdots \qquad \vdash f_n : i_n \to o_n}{\vdash (f_1, \ldots, f_n) : \sum_j^n i_j \to \sum_j^n o_j}$$

$$\textsc{Comp}\ \frac{\vdash f : i \to k \qquad \vdash g : k \to o}{\vdash (f : g) : i \to o}$$

$$\textsc{Pan}\ \frac{\vdash f : i \to k \qquad \vdash g : k * n \to o \qquad 0 < k \wedge 0 < n}{\vdash f <: g : i \to o}$$

Syntax and Typing

PL standard practice vs. what musicians want/imagine:

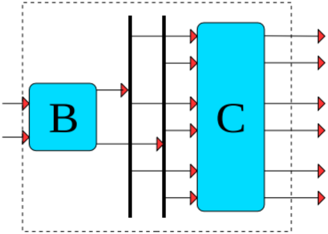


Figure 2: `(B:C)` sequential composition of B and C

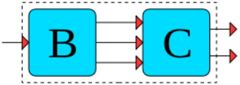
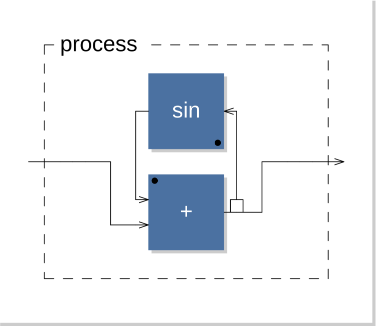


Figure 3: sequential composition of B and C when $k = 1$

Feedbacks

$$\text{FEED}\ \frac{\vdash f : g_o + f_i \to g_i + f_o \qquad \vdash g : g_i \to g_o}{\vdash f \sim g : f_i \to f_o}$$

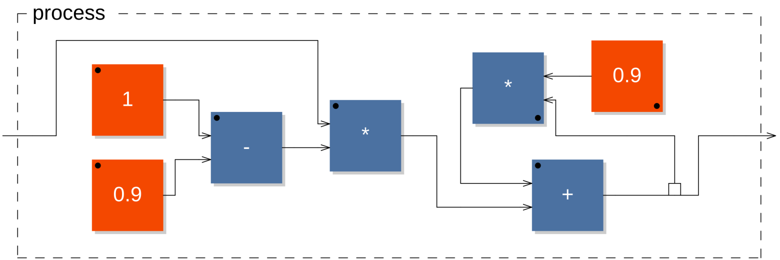
Diagram for `+` $\sim$ `sin`:



Synchronous semantics: execution in “ticks” + state.

Simple Low-pass Filter

$$\texttt{smooth}(c) = *(1-c) : + *(c);$$
$$\texttt{process} = \texttt{smooth}(0.9);$$



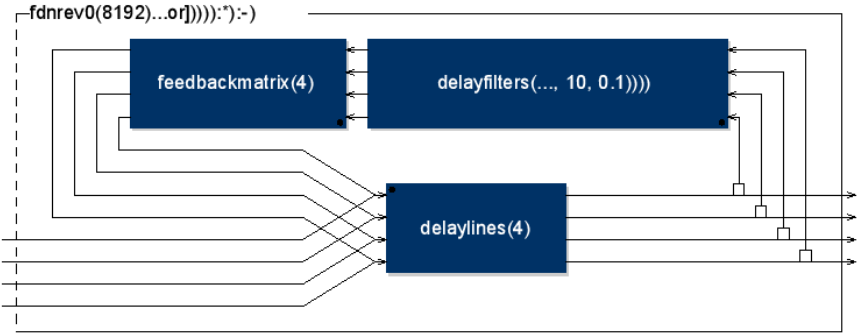
T:	1	2	3	4	5	6	7	8
I:	1.00	1.05	1.10	1.15	1.20	1.25	1.20	1.25
O:	0.10	0.19	0.28	0.37	0.45	0.53	0.61	0.68

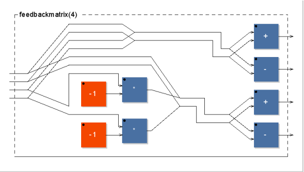
A More Real Example

Feedback Delay Networks:

```
fdnrev0(delays, BBSO, freqs, durs, loopgainmax, nonl)
  = (bus(2*N) :> bus(N) : delaylines(N))
    (delayfilters(N,freqs,durs) : feedbackmatrix(N))
with {
  delayval(i) = take(i+1,delays);
  delaylines(N) = par(i,N,(delay(dlmax(i),(delayval(i)−1))));
  delayfilters(N,freqs,durs) = par(i,N,filter(i,freqs,durs));
  feedbackmatrix(N) = bhadamard(N);
  vbutterfly(n) = bus(n) <: (bus(n):>bus(n/2)) , ...)
  ...
};
```

A More Real Example





Why Coq?

Why Coq?

Does there exist any other programming language?

Why Coq? Motivations and Goals

Philosophical

- Manual proofs starting to feel odd in PL.
- Motto: use Coq from the start.
- Goal: Try to develop in a reusable way.

Why Coq? Motivations and Goals

PHILOSOPHICAL — MATHEMATICAL

Current testing process of Faust programs: compare their output with MatLab's.

Why Coq? Motivations and Goals

Philosophical — Mathematical

Current testing process of Faust programs: compare their output with MatLab's.

- Program correctness.

Why Coq? Motivations and Goals

PHILOSOPHICAL — MATHEMATICAL

Current testing process of Faust programs: compare their output with MatLab's.

- Program correctness.
- Optimizations performed by the compiler are not well understood. *Semantics trickier than it looks to the eye*

Why Coq? Motivations and Goals

PHILOSOPHICAL — MATHEMATICAL

Current testing process of Faust programs: compare their output with MatLab's.

- Program correctness.
- Optimizations performed by the compiler are not well understood. *Semantics trickier than it looks to the eye*
- Explore the formalization of concepts from signal processing: Finite Impulse Response (FIR) filters, LTI theory, spectral analysis, Nyquist. . .

Why Coq? Motivations and Goals

PHILOSOPHICAL — MATHEMATICAL
PRACTICAL

Less effort than to build a custom analysis tool.

Applications:

Why Coq? Motivations and Goals

PHILOSOPHICAL — MATHEMATICAL
PRACTICAL

Less effort than to build a custom analysis tool.

Applications:

IMHO: **Robust Definitions and Standards** are crucial.

Don't repeat the mistakes of the past

Some Properties of Interest

- Stability properties: bound input produces bounded output. (We'll see an example)

Some Properties of Interest

- Stability properties: bound input produces bounded output. (We'll see an example)
- Linearity/Time invariance. [relational!]
- Stabilization: Zero input eventually produces zero output.
- Frequency response properties.

Some Properties of Interest

- Stability properties: bound input produces bounded output. (We'll see an example)
- Linearity/Time invariance. [relational!]
- Stabilization: Zero input eventually produces zero output.
- Frequency response properties.

In order to write the properties, we need a large support library

[bigops, intervals, trigonometry, Z-transforms, DTFT, . . .]

Specifications of Filters

Difference equations:

$$y(n) = x(n) + x(n-1)$$

Impulse response:

$$H(z) = \frac{1 - z^{-2}}{1 - 2R\cos(\Theta_c)z^{-1} + R^2 z^{-2}}$$

Two Poles Filter

$$H(z) = \frac{1 - z^{-2}}{1 - 2R\cos(\Theta_c)z^{-1} + R^2 z^{-2}}$$

```
process = firpart :+ feedback
with {
  bw = 100; fr = 1000; g = 1; // parameters - see caption
  SR = fconstant(int fSamplingFreq, <math.h>); // Faust fn
  pi = 4*atan(1.0);      // circumference over diameter
  R = exp(0-pi*bw/SR); // pole radius [0 required]
  A = 2*pi*fr/SR;        // pole angle (radians)
  RR = R*R;
  firpart(x) = (x - x'') * g * ((1-RR)/2);
  feedback(v) = 0 + 2*R*cos(A)*v - RR*v';
};
```

Finally! Let's Talk About Coq!

What we have built so far:

- Mathcomp allowed us to do a prototype in two weeks.
- New feedback reasoning rule & proof of soundness.
- *Motivated by real use cases*.
- Stateless logic & soundness (again, mc was key).
- Certified arity-checker, etc. . .

Finally! Let's Talk About Coq!

What we have built so far:

- Mathcomp allowed us to do a prototype in two weeks.
- New feedback reasoning rule & proof of soundness.
- *Motivated by real use cases*.
- Stateless logic & soundness (again, mc was key).
- Certified arity-checker, etc...

Currently:

- Investigating more complex, time-aware logics.
- New semantics based on guarded recursion.

The Pieces of the Puzzle

The First Piece: Stream-based Semantics

- We ported [Boulmé, Hamon and Pouzet], with some problems with CoInductives.
- We switched to sequences, (similar to Auger's Lustre certified compiler).
- Didn't look into PACO/more advanced co-reasoning tools.

The First Piece: Stream-based Semantics

- We ported [Boulmé, Hamon and Pouzet], with some problems with CoInductives.
- We switched to sequences, (similar to Auger's Lustre certified compiler).
- Didn't look into PACO/more advanced co-reasoning tools.

Current approach: realizability semantics in guarded recursion style. Suggested simultaneously by A. Spiwak and A. Guatto [SYNCHRON 2014]:

$$[\![\vdash f : i \to o]\!]^n_W : [\![i]\!]^n \to [\![o]\!]^n$$

The Second Piece: Real Analysis

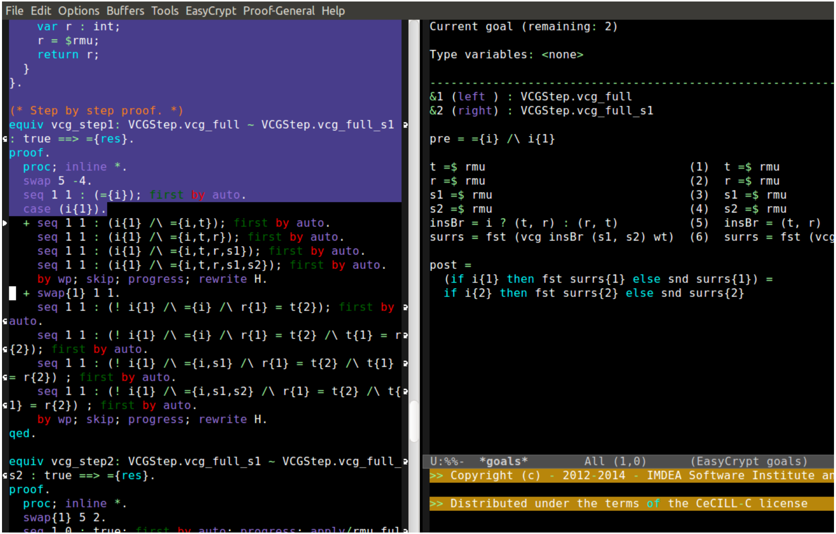
- Not in Mathcomp – rcf good enough for experiments.
- Our typical case involves complex numbers, trigonometry and sums over infinite series.
- Think of proving Euler's formula:

$$e^{(j\Theta)} = \sin\Theta + j\cos\Theta$$

- Difficult to choose: Standard library? Coquelicot? C-CorN?
- Our feeling is that life is going to be very painful.

[We are ignoring floating point issues for now]

The Third Piece: Coq as a Tool



The Third Piece: Coq as a Tool

- Is building a verification tool on top of Coq feasible?
- We got some inspiration from domain-specific tools like EasyCrypt.
- Would our tool mature, we would certainly need to plug deeply into Coq's parsing/display routines.
- We still think this may be better than rewriting everything from scratch.
- Reduction woes make our life difficult.
- Automation: we will worry last.

Stability of Smooth

Recall the smooth program:

```
smooth(c) = *(1−c) : +  *( c);
```

We want to prove stability, that is, bounded inputs produce bounded outputs, provided the coefficient c is in $[0, 1]$.
We use the following logical rule (simplified):

$$\frac{\models \psi(x_0) \qquad \{\gamma(i_1) \wedge \phi(i_2)\}\ \ f\ \ \{\psi(o)\} \qquad \{\psi(i)\}\ \ g\ \ \{\gamma(o)\}}{\{\phi(i)\}\ \ f \sim g\ \ \{\psi(o)\}}$$

Stability of Smooth

Three VC in the proof:

```
by rewrite ?ler_wpmul2r ?ler_subr_addr ?add0r.

have Ha: a = a * c + a * (1 - c)
  by rewrite -mulrDr addrC addrNK mulr1.
have Hb: b = b * c + b * (1 - c)
  by rewrite -mulrDr addrC addrNK mulr1.
by rewrite Ha Hb !ler_add.

by rewrite ?ler_wpmul2r.
```

We pushed the VCs to Why3 with success.
Interval technique ready to go into the main compiler.

Conclusions

- Young project, highly positive experience so far.
- First alpha “release” very near.
- Tons of related work, difficult to get a good perspective.
- Most challenging topic: real/complex analysis.
- Certified audio/dsp processing? (Do we need it?)
- All of the usual Coq caveats apply to us.
- What do *you* think?

Thanks!

Nyquist Theorem

Provided f_s is twice the highest frequency in V then:

$$V(t) = \sum_{n=-\infty}^{\infty} V[n] \frac{\sin[\pi f_s (t - nT_s)]}{\pi f_s (t - nT_s)}$$

where

f_s	$= 1/T_s$	sampling frequency
$V(t)$		value of signal at Time t
$V[n] = V(nT_s)$		value of signal at Time $t = nT_s$